Cargo Data Miner User Guide

Qin, Xuzhou

Under the tutorship of
Von-Tronchin, Oliver and Binet, Pierre

August 2016

Contents

Abstract

This is the user guide of Cargo Data Miner. Cargo Data Miner is an R-Shiny Dashboard application developed by the Airbus Freighter Marketing department (CSMX). The initiative is to build a platform where everybody could query and analyze cargo data. In this document, detailed information and instruction are provided in order to facilitate the usage and maintenance of the application. For more information, the author could be reached at xuzhou.qin@gmail.com.

Acknowledgement

The present work benefited from Oliver Von-Tronchin, tutor of my internship and director of Freighter Marketing (CSMX), Sebastian Aponte, Nicolas Chauviere-Courcol, colleagues of mine in CSMX, Federico Semeraro and Andy Williams, for their inspiration of the project *LiMA*, Pierre Binet, Didier Gauzy, Aurelien Turina, Sam Crawshay Jones, Lionel Cousseins, Yves Renard and all my colleagues of Airbus who provided valuable comments and ideas to the undertaking of the project here.

1 Introduction

Aviation market is a highly complex and data-dependent market. Inside Airbus, numbers of different servers are deployed to stock, manage and provide database services to support our daily work. It is striving to introduce data digitalization as a core task in the future. The increasing importance of data has already grabbed the attention of the managers and will never stop growing. However, the massive scale of data brings more challenges than solutions.

At the company level, the main challenge comes from database management. Too many databases are stocked seperately with different **database management systems** (DBMS). The databases are quite isolated from one another. In other words, except the people who frequently use one database, others would not even know its existence. Moreover the different DBMSs cause inconvinience in database management. For example in the freighter marketing department (CSMX), three cargo marketing databases, *Cargo IS*, *Seabury* and *FlightRadar 24* are stocked separately in three data servers using different DBMSs. However the freighter marketing department is not the only one that suffers from this problem.

At the department level, the major challenge is the lack of an efficient and user-friendly tool for the data extraction and analysis process. Even though the traditional approach of database management, SQL[1], is widely used for data manipulation. In order to use it efficiently, you must have some knowledge of programming and you have to cope with the difficulty in interfacing. As SQL is not a common tool inside marketing department, interacting with the data by using SQL may not be a suitable choice. As to the data analysis tool, Excel is one of the most widly used software. But it suffers from some innate limitations facing the increasing amount of data.

There are also some Airbus-made tools for the marketing departments, like *Acper*, *OAG Builder*, *route06*, etc. These are very powerful and sophisticated programs dedicated for specific usage thats existed for a very long time. They are irreplaceable for some tasks but they are not perfect. For example, they both have a steep learning curve. Some special training session are required to master these tools. From a graphical design's point of view, they often remind me of the "old fashioned" software interface, which may be not intuitive and user-friendly. It is essential to visualize the analytic results in a modern way.

Due to these existing shortcomings, data digitalization inside Airbus could be done from two levelshould target two aspects. We will need an uniform and standardized platform to stock and manage data at the company level; and inside the departments, new data mining and analysis tools are needed to support the work in the Digital Age. Some projects have already been moving toward that direction inside Airbus. For example, we developed **BIO**, which is a platform that provides an automatic aggregation of different databases and **Foundry**, which is a cloud-based platform for collaborative big data analysis built by Palantir Technologies.

The objectif of my internship is thus to find an appropriate solution that fits the data analysis request of the Freight Marketing department. A new tool is needed to replace the traditional data extraction procedure, which is repetitive and time-consuming, sometimes even impossible due to the large volume of data. Meanwhile, a new data visualization approach is also needed in order to communicate marketing information more clearly and efficiently. Therefore, I will use my knowledge and experience learnt at school to help increasing the productivity of CSMX.

[1]Structured Query Language, which is a language designed for managing data in a ralational database management system.

This document provides a feasible solution for the development of a data analysis tool by using **R Shiny Dashboard**. **R Shiny Dashboard** is a framework to create a light weight, powerful and interactive web application. It uses a reactive programming model so that outputs change instantly as users modify inputs. It is easy to use and no web designing language required. Further customization could be easily done with some HTML, CSS and JavaScript[2] knowledge.

In the first section, I will explain the reason for which **CDM** was born. After that, the installation and deployment prerequisites are presented. In section four, detailed functionalities and the application structure, as well as the data structure, are stated in order to provide technical support for using and maintenance. In the fifth section, some suggestions for the future development of this application are proposed. The technical appendix could be found in the last section of the document.

2 Why R shiny Dashboard?

At the beginning of my internship, I have been asked to explore different cargo databases. Because of my statistical background, I was using R to analyze and visualize these data. Lots of codes were written at that moment, as well as numbers of power-point slides. But my tutor and I realized that coding in R could not be a permanent solution for the needs of Freight Marketing (CSMX) team, because R is not a common tool inside marketing department and it is not that easy to learn for people with little programming knowledge.

I started looking for a better way to make a reusable, easy-to-access yet powerful data analytic tool. I learnt a lot about R at school so my first intention is to keep using R as the programming language. Furthermore, the number of R users at Airbus is increasing steady. It could be a big advantage for the further development of the code.

Shiny by Rstudio is a great tool that fits our needs. Shiny offers the *reactive programming* framework that perfectly matches the need of a reusable tool. It allows us to produce different analysis from different inputs, without modifying the code. Meanwhile, the abundant R packages could make life much easier by dramatically reducing the programming workload.

Moreover, by using Shiny Dashboard, it is easy to integrate different shiny applications into a one-page web application. So that we will have one complex dashboard based on the **Bootstrap 3**[3] framework. It is intuitive, efficient and multifunctional

It has been four months since I started to develop **CDM**. Now the basic framework is almost done. In the future, I hope it could help us work more efficiently.

3 Deployment guide

Cargo Data Miner (CDM) was developed with **Microsoft R Open 3.2.5**, which is an enhanced distribution of R. It supports by default multi-thread computing and it is perfectly compatible with the original **R**. The user interface used during the development is **RStudio**, which the the most common IDE of R.

To make **CMD** work properly, there are two main methods.

[2]HTML, CSS and JavaScript are the most common languages dedicated for the modern web development.
[3]Bootstrap 3 is a HTML, CSS and JS framework for developing responsive, mobile first projects on the web.

– Install and run it locally on a PC for personal use.

– Deploy it on a Shiny server so that CDM could be reached by an IP address.

In this section, I will present both of the methods.

3.1 Directory Structure

The following figure shows the structure of the CDM directory.

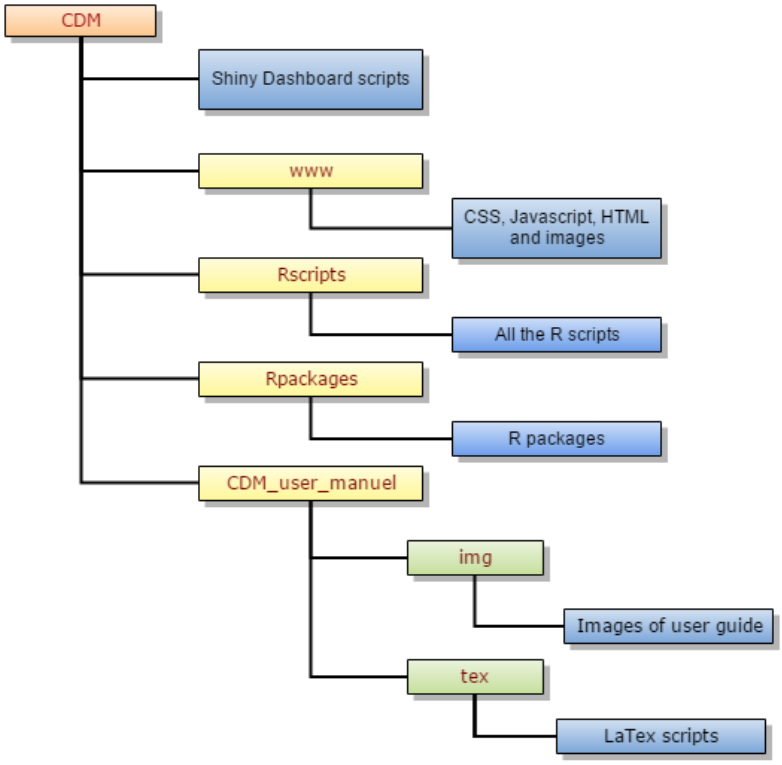

Figure 1: CDM directory structure

The main directory `~/CDM` contains Shiny Dashboard scripts (*ui.R*, *server.R* and *global.R*) and three directories:

- `/www`
- `/Rscripts`
- `/Rpackages`
- `/CDM_user_manuel`

All the images, HTML, CSS and JavaScript used in **CDM** should be placed in `~/CDM/www`. All the R scripts called by the application should be placed in `~/CDM/Rscripts` and all the R packages (for

local installation) should be placed in `~/CDM/Rpackages`.These three directories are the essential parts of **CDM**.

In `~/CDM/CDM_user_manuel`, you will find the code of this use guide (written in R markdown) and its resources. All the images that appear in this document should be placed in `~/CDM/CDM_user_manuel/img` and all the LaTex scripts called by R markdown should be put in `~/CDM/CDM_user_manuel/tex`.

3.2 Installation on a PC

CDM could run on a personal computer with an installed R and a web browser with the correct configuration. Before the first launch, some steps have to be followed.

- Step 1: Get R and other required software:
  - Install **Microsoft R Open 3.2.5** from https://mran.revolutionanalytics.com/download/ (to active the multi-thread computing support, the `MLK` package should be installed as well), or you can install the original **R** (64-bit) from https://cran.r-project.org/mirrors.html to maximize the reliability.
  - Install **RStuido** from https://www.rstudio.com/products/rstudio/. This is optional but highly recommended.
  - **CDM** works with Internet Explorer but you could also install Google Chrome for the best performance.
  - Install **Oracle Database Instant Client (11.2.0.3) EN_W764** from PC service. Or you can download it from the Oracle website (Make sure that the downloaded software is the 64-bit version). If you get errors about Oracle client during the execution of **CMD**, see Appendix: Troubleshooting for help.
- Step 2: Connection Setup
  - **CMD** needs the access right of three different databases, which are: **p595dodmp01** (Cargo IS, Seabury), **fr0-bio-p01** (BIO) and **fr0-dmds-p01** (FlightRadar 24). User names and passwords are already pre-filled and there is no need to change. Except for **BIO** database, you will need to grant the access right to your own *Windows* account to establish the connection. To do so, you could ask *Aurelien Turina* (aurelien.turina@airbus.com) or *Pierre Binet* (pierre.binet@airbus.com) for further information.
  - `~/CDM/server.R` contains the connection parameters of these databases.

```
############################################################
#' CONNECTION PARAMETERS
#'
#' Modify it before the first execution of the application
############################################################
#' ROracle connection string
#' for Cargo IS, Seabury
host = 'p595dodmp01'
```

```
port = 1521
sid = 'DBUPA269'
username_cargois <- 'CARGO'
password_cargois <- '********'


#' RODBC MS SQL server connection string
#' for BIO
driver_bio <- 'SQL Server'
server_bio <- 'fr0-bio-p01'
port_bio <- 10335
username_bio <- ''
password_bio <- ''
trusted_connection_bio <- TRUE # if TRUE connect with windows login and pw


#' RMySQL MySQL connection string
#' for FlightRadar 24
user_fr24 <- "user_ext"
password_fr24 <- "************"
dbname_fr24 <- 'FR24'
host_fr24 <- "fr0-dmds-p01"
```

- Step 3: Now you can open RStudio.
  - Before running **CDM**, if you have installed more than one version of R on your computer, make sure that the R version is `[64-bit] ~/MRO_3.2.5` (the name could be varied)
  - Click on the **run** button on the top right of the text editor aera to execute the application. **CDM** will automatically check the required R packages and it will install from **CRAN** (Comprehensive R Archive Network) those which are missing on your computer . **CRAN** is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. However, due to the internet connection configuration inside Airbus, if the installation of the required packages fails, you will have to download and install it manually (See Appendix for instruction).

When the interface shows in your web browser and the progress indicator disappears, you are finally ready to go :)

3.3 Deployment on a Shiny Server Pro

Shiny Server Pro is a server program that makes Shiny application available over the web.

At this time, Shiny Server can be run on Linux servers with explicit support for Ubuntu 12.04 or greater (64 bit) and CentOS/RHEL 5 (64 bit) or greater. Multiple Shiny applications on multiple web pages could be hosted with the same Shiny Server.

It can be downloaded from https://www.rstudio.com/products/shiny/shiny-server/.

For the detail instruction of the configuration and installation of Shiny Server Pro, see **Shiny Server Professional v1.4.2 Administrator's Guide** on http://docs.rstudio.com/shiny-server/.

4 Structure and Features

A Shiny dashboard application could contains three sources scripts:

- a user-interface script: `ui.R`
- a server script: `server.R`
- a global environment script: `global.R`

There is another structure of Shiny application where all the three main scripts could be integrated into one single script `app.R`. But here we only talk about the "traditional" structure.

The UI script controls the layout and appearance of the app. The server script contains the instructions that your computer needs to build your app. And the global script defines objects in the global scope.

Reactivie programming is the fundamental of a Shiny Dashboard application. The main idea is to make it easy to wire up *input values* from a web page, trigger the R code to be (re)executed, and then have the results of your R code be written as *output values* back to the web page.

The following figure shows the three elements of Shiny reactive programming. **Reactive source** typically is the user's input through a web browser. It could trigger the execution of an R code. The output of the code is represented by the **reactive endpoint**. **Reactive conductor** could be placed between the sources and endpoints. It could be used to store output so that other function could directly use it instead of redo the computing. Generally they are used to encapsulate the computationally expensive operations in order to augment the code performance.

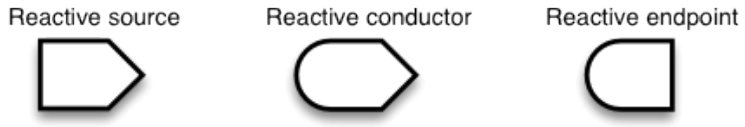

Figure 2: Essential objects in reactive programming

After launching **CDM**, you will see the following welcome page. **CDM** contains three parts: a header, a sidebar and a body. The header shows the application name and notification[4], for example the database connection status, the app version and the developing progress. The sidebar could help you to navigate between different tabs. The functionalities of these tabs will be presented in the following sections. Finally the body is the main area where you interact with **CMD** and get the output.

Five different tabs are displayed on the sidebar. Each tab contains also several sub-tabs.

[4]Some features are still under development.

Figure 3: **CMD** home page

4.1 Cargo IS Database

Cargo IS database is the first major component of **CDM**. The database is stored on the server **p595dodmp01**, under the user folder **A269_CARGOIS**. The main data table is `CARGOIS_MONTHLYDATA`, which contains 33 columns. But only few of them contain usable information for analysis.

Cargo IS database provides monthly Air Waybill data in the airport level, It contains airfreight charges, cargo weight and number of shipments (grouped by different weight breaks[5]). In Cargo IS, the AWBs are regrouped by their weight, we have thus six categories of AWB (detailed explanation can be found in section *Air Waybill Analysis*).

Cargo IS is the only resource of *cargo* yield information for Airbus.

Remark:

- The database that we purchased does not contain any airline information. However we could find a column called `AIRLINE_PREFIX` in the data table. It means that IATA do have airline data but this information is excluded in our database. It may be possible to negotiate with IATA to get the airline information in the future.
- The data of Cargo IS was collected from airlines. However, it covers only 60 countries of the world, which means sometimes have to take into account the data representativity. Some important countries are missing, for example, India, vietnam, russia and almost all the african countries.(see Appendix II for detail)
- The are 25 customizable weight breakes (provided by IATA). Furthur study could be focused on how to get the best weight breaks configuration, which could provide more information.

[5]Weight breakes are the break points that split the countinuous cargo weight variable into different categories, *e.g.* 0-45kg, 45-100kg, *etc.*

4.1.1 Data loader

4.1.1.1 Instruction

The first step is about how to load Cargo IS data. In the first drop-down menu (*Query level*), we could select the level of data granularity to be queried (*e.g.*, airport to airport, country to country, region to region, etc). Then the year and the **O&D** (**O**rigin and **D**estination) of airfreight should be precised. You can check the box *"Plot evolution since 2010"* to load data of all the available year (2010-now). When the box is checked, the year input will be ignored.

After selecting the right parameters, click on **Load Data** button. Some summarize of the queried dataset will be shown on the top of this tab. A preview of the 50 first rows will appear on the right. You could do a quick check of the data structure. The dataset could be downloaded by clicking on the **Download Data** button.

Now the dataset is loaded and ready to be used!

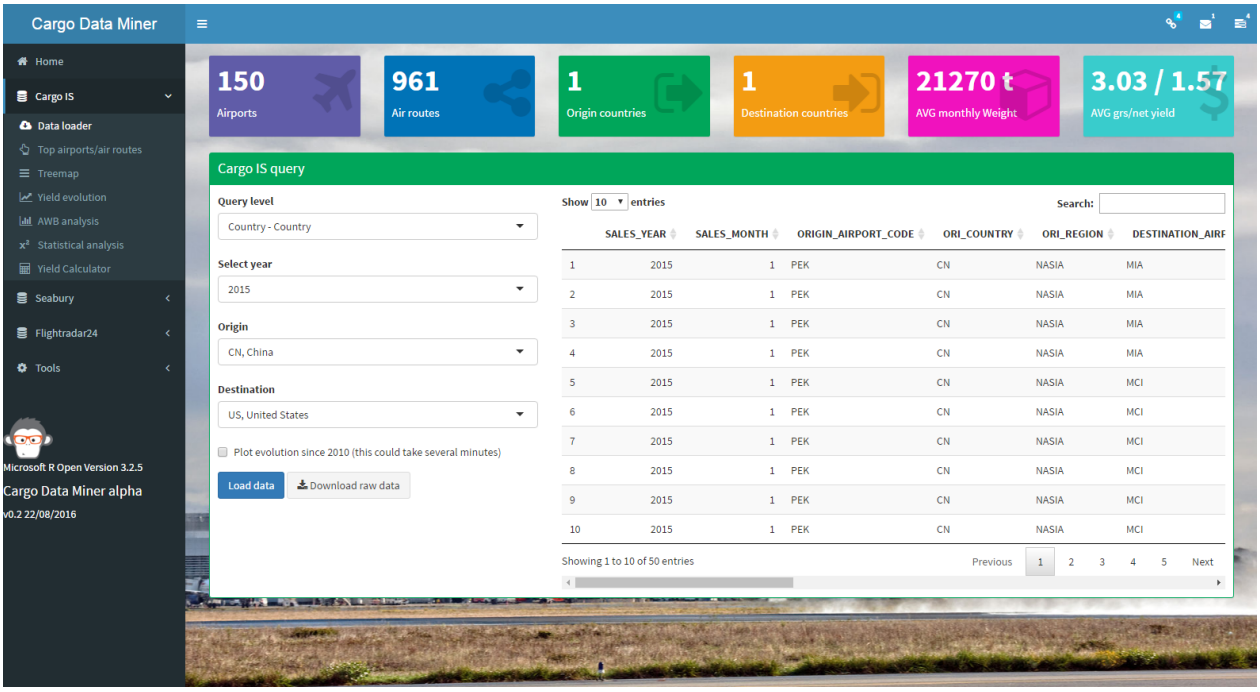

Figure 4: An example query of the 2015 airfreight from *China* to *the USA*.

4.1.1.2 Explanation

The data loading procedure is done by calling the user-defined function `CargoIS.SQL.query()`. The function will take the user parameters as inputs, generate an appropriate `Oracle SQL` query and send it to the data server. The returned dataset is thus stocked in the global scope of Shiny and it is ready to be used by other functions.

```
# EXAMPLE
# Query airfreight data of 2015 with origin country = China, destination = USA
dat <- CargoIS.SQL.query(CargoDB, level = 'C2C',
```

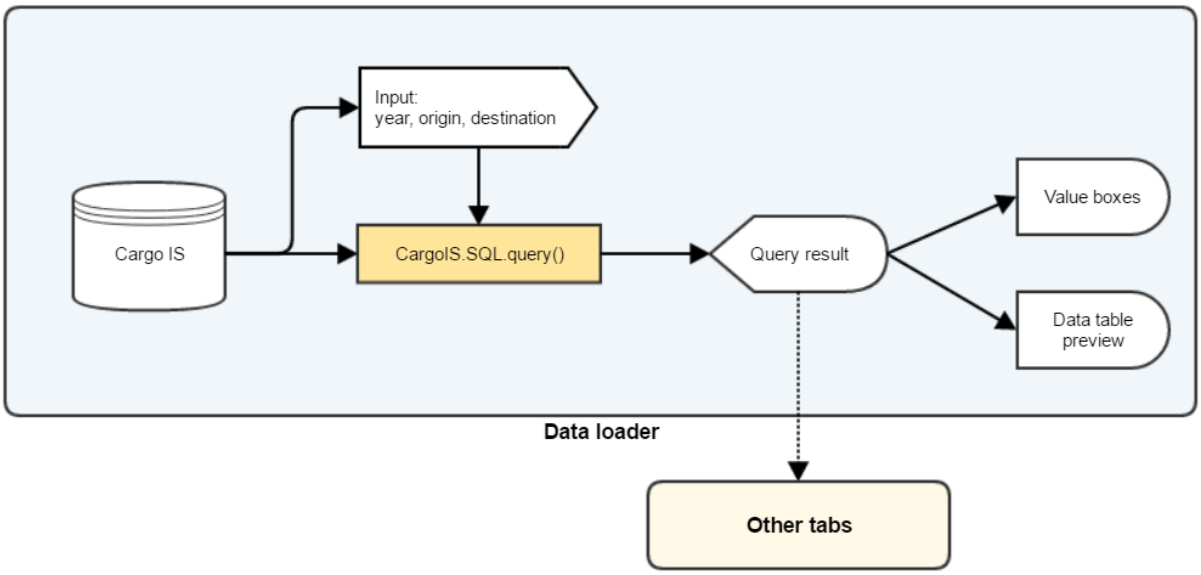

Figure 5: Flowchart of data loading process

```
                ORG = 'CN',
                DST = 'US',
                year = 2015)
```

The six value boxes on the top of the page are created by the function `valueBox()` based on the output dataset. They are:

- **Airports**: the number of unique airports appeared in the dataset.
- **Air routes**: The number of unique airport pairs (Origin + Destination) in the dataset.
- **Origin country**: The number of unique origin country(ies).
- **Destination country**: The number of unique destination country(ies).
- **AVG monthly weight**: Arithmetic mean of weight.

$$Average\ monthly\ weight = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} weight_{ij}}{n}$$

, where i is the number of months, j is the weight break.

- **AVG gross/net yield**:

$$Average\ gross\ yield = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} charges_{ij} + surcharges_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{m} weight_{ij}}$$

$$Average\ net\ yield = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} charges_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{m} weight_{ij}}$$

, *charges* are fees directly related to the weight of the shipment and *surcharges* contain all the other fees (fuel surcharges, security surcharges, taxes, etc.)

4.1.2 Top airports/air routes

When the data loading procedure is complete, you can go to the following tabs. After clicking on the "Top airports/air routes" tab, **CDM** will automatically proceed and visualize the loaded dataset.

4.1.2.1 Instruction

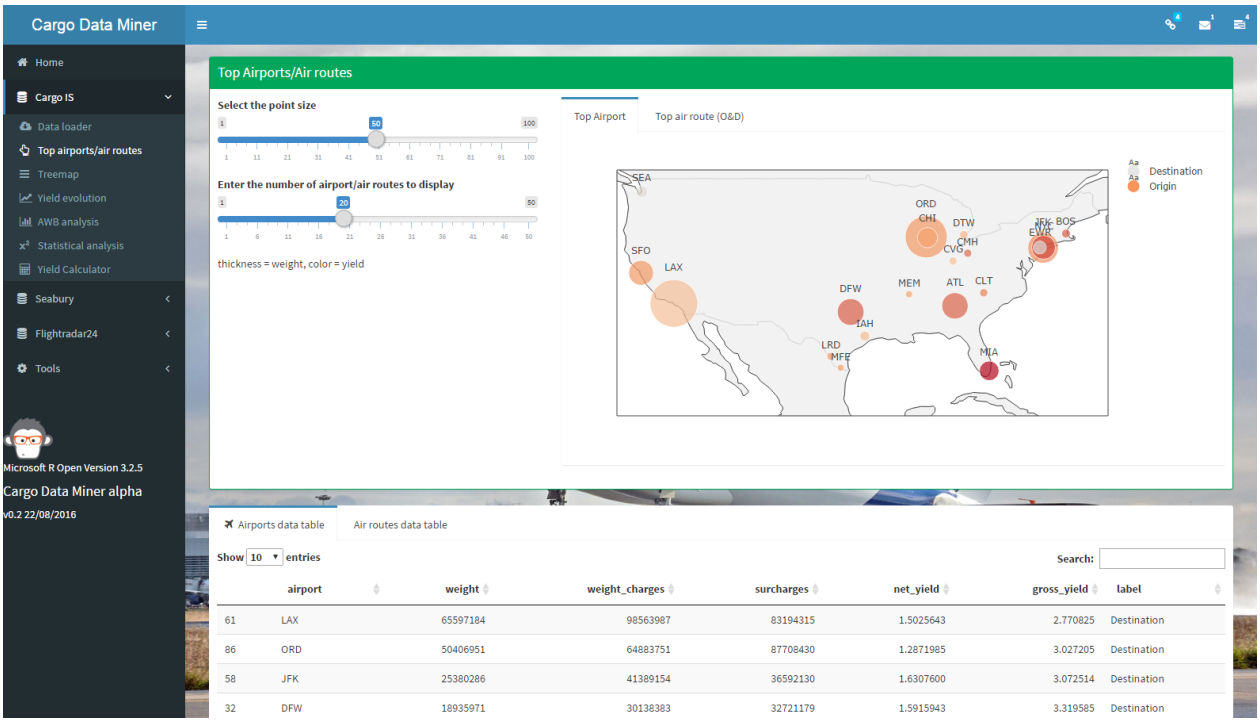

Figure 6: Yield Analysis tab, top 20 airports of CN-US

The first part of tab contains visualization of the **top *n* airports and air routes**. Figure 6 shows in the map the top 20 origin and destination airports. The size of the circle represents the weight of shipment going through this airport. The darkness of the color represents the yield of the airport. The charts are interactive, you can zoom in/out with the mouse scroll wheel. You can also click on the legend to display/hide the origin/destination airports.

The value of n could be modified by dragging the slider bar. You could also modify the size of the circle if sometimes the default size is not optimal.

The **Top air routes (O&D)** tab of the visualization chart shows the weight and yield of the most important air routes. Origin and destination airports are linked by blue curves with different thickness and darkness. The **thicker** the curve, the **more** the total weight and the **darker** the blue, the **higher** the yield.

The last two tabs (Airports data table and Air routes data table)show the two data tables that **CDM** uses to create the map. It could be sorted by variables and it could be directly copy-pasted into an Excel sheet.

4.1.2.2 Explanation

An R package `Plot_ly` was used to create the visualization map. From `/Scripts/Plot.airport.html.R`, an user defined function `Plot.airport.html()` is called.

```
#' [Plot.airport.html] could be used to visualize the top n airports/air routes.
#' When the parameter table = TRUE, it will output the datatable that it
#' uses for the data visualization.
#'
#' @param dataset: data frame with five columns
#' c('ORIGIN_AIRPORT_CODE', 'DESTINATION_AIRPORT_CODE', 'WEIGHT_CURRENT_YEAR',
#' 'WEIGHT_CHARGES_CURR_YEAR_USD', 'OTHER_CHARGES_CURR_YEAR_USD')
#'
#' @param projection: Mercator or orthographic
#'
#' @param top: numbers of airport/air routes to be displayed
#'
#' @param plot.airport: if TRUE, plot airport, if FALSE, plot air routes
#'
#' @param pt.size: int, point size
#'
#' @param table: if TRUE, output datatable, otherwise, output map

 Plot.airport.html(dataset, projection = 'Mercator', top = 20, plot.airport = TRUE,
                   pt.size = 800000, table = FALSE)
```

`Plot.airport.html()` will call `plotly::plot_ly()`. This function provides a simple way to add scatters/lines/curves to a map. Only the coordinates of the two airports are needed to draw the air route.

The coordinates of airport can be acquired from BIO database. The user-defined function `get.airport.loc()` could be used to get the airport geographical information.

```
source('/Scripts/get.airport.loc.R')
#' When detail = FALSE, it will return airport code, longitude and latitide
#' if detail = TRUE, it well return city name, country name, region names
#' of the airport
get.airport.loc(detail = FALSE)
```

Figure 7 shows how the function `plot.airport.html()` works.

4.1.3 Treemap

4.1.3.1 Instruction

The tab **Treemap** visualizes the characteristics of **nodes**[6] (airports, countries and regions). Each node is displayed as a rectangle, sized by the total airfreight weight of this node and colored by its average gross yield.

You can click on the rectangle to see its child nodes[7] (if exist), and right-click to see its parent node.

[6]Nodes are data points that represent the information contained in a single structure.

[7]For example, **country** is the child node of **region**, and it is the parent node of **airport**.

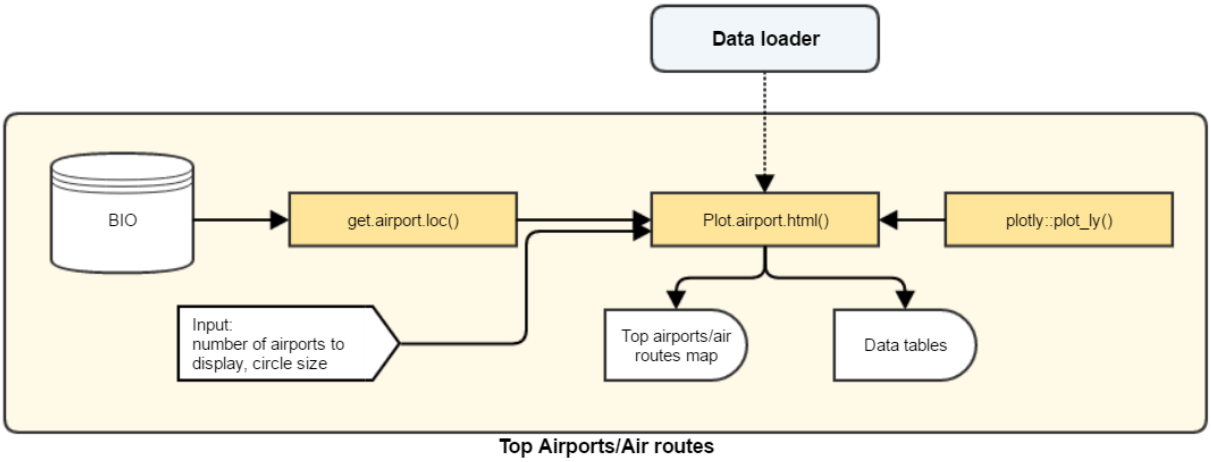

Figure 7: Flowchart of top airports/air routes tab

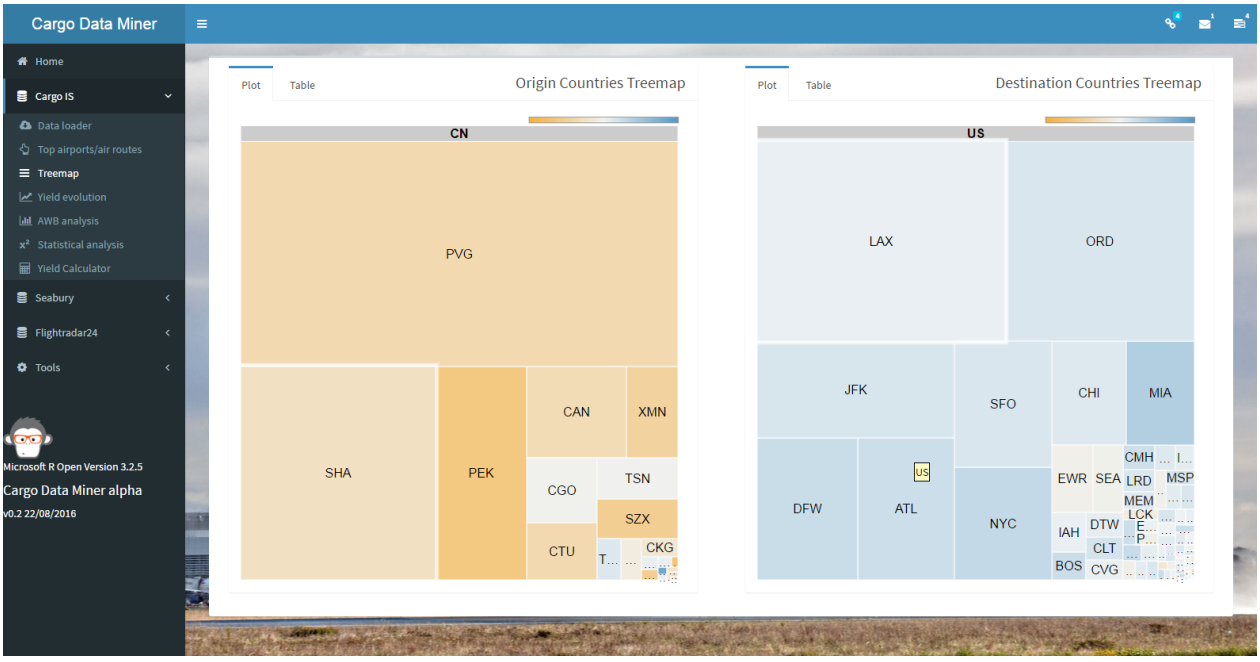

Figure 8: Treemap of airports for CN and US

A scale is shown on the top right to indicate the yield level. The tab **Table** will render the data table that **CDM** uses for the visualization. The table contains four columns, *Name*, *Parent*, *Weight*, *Gross Yield*.

Currently there is no optimal solution to generate a static downloadable image. So you may take a screen shot if you want use it for somewhere else.

IMPORTANT: The color of treemap is not horizontally comparable, which means that you can not compare the yield of an origin airport with the yield of a destination airport.

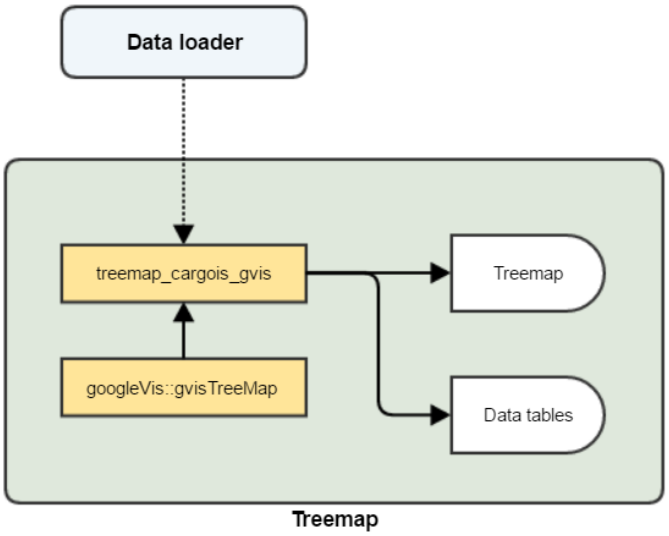

Figure 9: Flowchart of Treemap tab

4.1.3.2 Explanation

To create an interactive treemap, **CDM** uses the user-defined function `treemap_cargois_gvis()`, which will call `googleVis::gvisTreeMap()`. The package `googleVis` is the R interface to **Google Charts API**. It generates an HTML code which could be integrated into R Shiny application. It could be further customized by using Javascripts.

First of all, the function `treemap_cargois_gvis()` will convert the queried Cargo IS data table into a four-column dataset (*Name*, *Parent*, *Weight*, *GrossYield*), which could be used by `googleVis::gvisTreeMap()`.

```
# Example:
treemap_cargois_gvis(data, ORG = FALSE, PLOT = TRUE)

#' data is a data.frame object generated by Cargois.SQL.query()
#' If ORG = TRUE, then create treemap for the origin airports,
#' else for the the destination airports
#' If PLOT = FALSE, then the output will be a data.frame object
#' (instead of a plot)
```

Then `gvisTreeMap()` will generate the treemap.

```
# Example of gvisTreemap
gvisTreeMap(data = datatable, idvar = 'Name', parentvar = 'Parent', sizevar = 'Weight',
                colorvar = 'GrossYield',
                options=list(fontSize=16,
                             minColor='#F5B041',
                             midColor='#F0F3F4',
                             maxColor='#5499C7',
```

```
                        headerHeight=20,
                        fontColor='black',
                        showScale=TRUE)
                    # options, see
                    # https://developers.google.com/chart/interactive/docs/
        )
```

4.1.4 Yield Evolution

Yield is the most important information in Cargo IS database. It shows the profitability of air routes. The evolution of yield indicates the economic situation of the market. #### Instruction

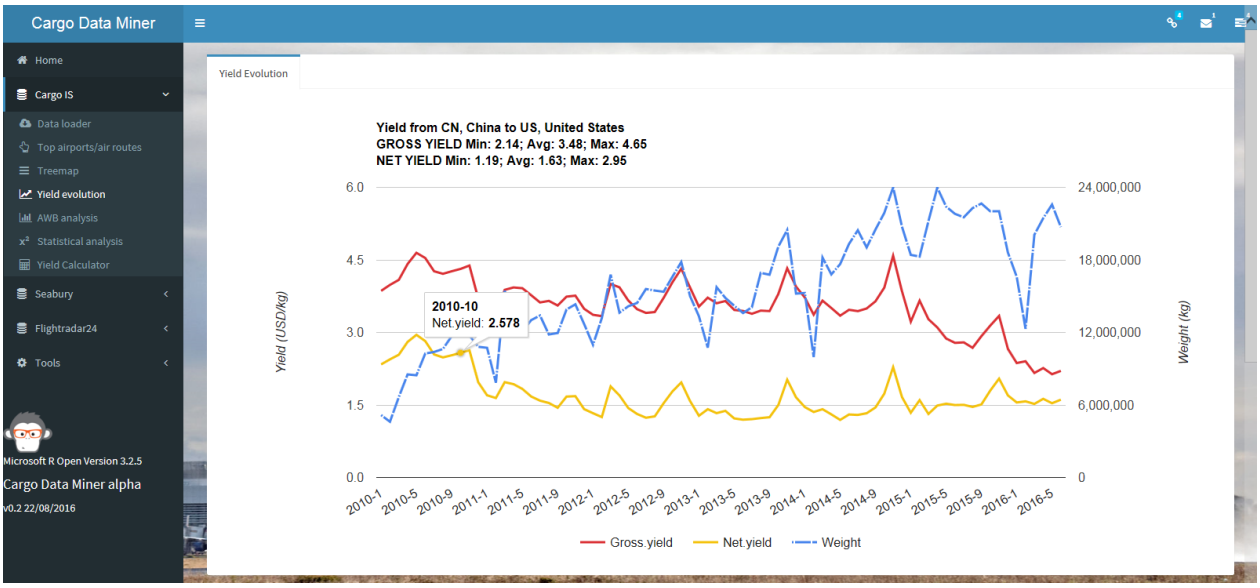

Figure 10: Airfreight weight and yield evolution for CN and US

In the next tab **Yield evolution**, an airfreight evolution line chart will be displayed. **CDM** calculates and render a line chart showing the evolution of gross yield, net yield and airfreight weight during the selected period of time. The minimum, maximum and average value of the output are shown in the plot's subtitle.

The x-axis represents the timestamps during the selected period. The y-axis shows the yield (USD/kg) on the left and the weight (kg) on the right.

If the box "Plot evolution since 2010" in the data loaded page is checked, all the available data will be visualized (but the computing will also take a longer time).

You could click on the curve to see detailed information of the chosen data point.

A data table appears on the bottom of the page. It contains all the data of the line chart. You could click on the download button to save it to the local.

4.1.4.1 Explanation

I would like to integrate the three curves (net yield, gross yield and weight) into one single chart. Thus two different y-axis are needed. The well known package `ggplot2` does not support two different y-axis in one chart so I will still use `googleVis` to generate the line chart and there is still no appropriate way to generate a static image to be downloaded.

```
# Example of givsLineChart
# with gvisLineChart, we could easily create a line chart with 2 y-axis

gvisLineChart(data, xvar = "xaxis", yvar = c('yaxis_1', 'yaxis_2', 'yaxis_3'),
              options = list(series="[
                                {targetAxisIndex: 0}, # set the target y-axis
                                {targetAxisIndex: 0},
                                {targetAxisIndex: 1}
                                ]"
                             )
              )
```

4.1.5 Air Waybill Analysis

An **air waybill** is a receipt issued by an international airline for goods and an evidence of the contract of carriage. It is the most important document issued by the carrier (directly or through an authorized agent). Cargo IS contains weight data and it regroups the air waybills according to their weight into six categories, which are called **weight breaks**.

They are:

- 0 to 50 kg
- 50 to 100 kg
- 100 to 300 kg
- 300 to 500 kg
- 500 to 1000 kg
- larger than 1000 kg

The proportion of the AWB of different categories in a market could be an indicator of the economic situation. For example, an increasing number of heavy shipments often means the cargo consolidation becomes more efficient, and the economic situation is going better.

4.1.5.1 Instruction

The first window of this tab contains seven figures:

- *Shipment frequency evolution* is a line chart which shows how the numbers of shipments change during the selected period. By default, there will be six curves representing respectively the different weight break.
- *Shipment weight distribution* is a bar chart which shows the distribution of the AWB of different weight break in the selected period.

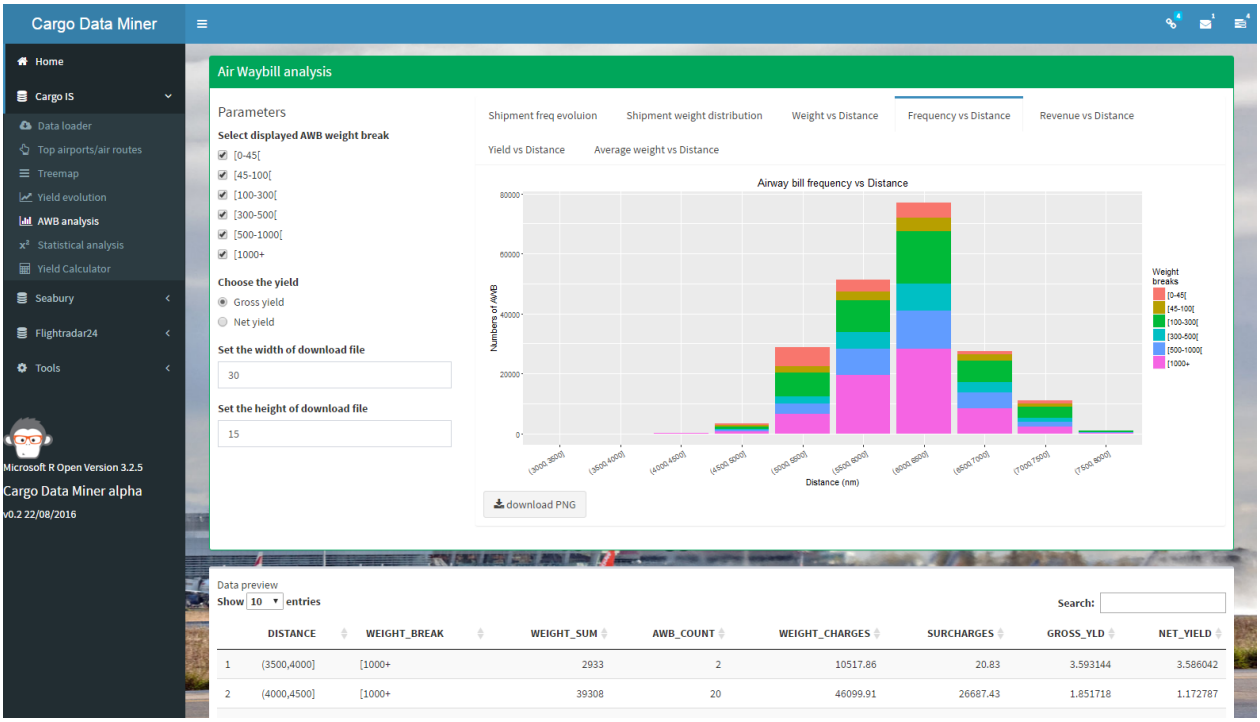

Figure 11: Air waybill analysis

- *Weight vs. distance* is a stacked bar chart showing the sum of shipment weight by different intervals of distance. The colors represent the different weight breaks.
- *Frequency vs. distance*: like the previous figure, it shows the numbers of shipments by different intervals of distance.
- *Revenu vs. distance* shows the revenues (charges) of airfreight by different intervals of distance.
- *Yield vs. distance* is a line chart showing the gross/net yield of by weight breaks and by distance.
- *Average weight vs. distance* takes into account all the weight breaks and compute the weighted arithmetic mean of weight by distance.

For the charts 3, 4 ,5 and 6, if there is only one interval of distance, a pie chart will be created instead of a bar chart. All the charts can be downloaded by clicking "download PNG" button.

On the left of the charts, some parameters can be changed to modify the charts. You can display/hide one or multiple weight break(s) by checking/unchecking the boxes. The second part "gross yield/net yield" corresponds to the yield type in the chart *Yield vs. distance*. The last two text input area control the width and height of the downloaded image.

On the bottom of the page, the data table used for creating the charts is displayed. It could also be downloaded by clicking the download button.

4.1.5.2 Explanation

Before creating the charts, data manipulation should be done in order to have the right data structure. `data.table` is a very powerful package for big data computing.

```
# Example of data.table

system.time(
  output <- dt[,.(weight = sum(WEIGHT_CURRENT_YEAR),
                  freq = sum(AWB_COUNT_CURRENT_YEAR),
                  wchg = sum(WEIGHT_CHARGES_CURR_YEAR_USD),
                  surchg = sum(OTHER_CHARGES_CURR_YEAR_USD),
                  gy = (sum(WEIGHT_CHARGES_CURR_YEAR_USD)+
                            sum(OTHER_CHARGES_CURR_YEAR_USD))/sum(WEIGHT_CURRENT_YEAR),
                  ny = (sum(WEIGHT_CHARGES_CURR_YEAR_USD))/sum(WEIGHT_CURRENT_YEAR)
                  ), by = .(dist = lebal, wb = WEIGHT_BREAK)
                ]
)
# dt is a 1 750 000*15 data.table,
# user  system elapsed
# 0.14    0.00    0.14
```

To create these charts, the package `ggplot2` is extensively used.

4.1.6 Yield Calculator

The last sub tab of Cargo IS is a marketing tool called yield calculator, which offers an automatic computing for air route cargo yield. It will read air routes from a `csv` file (uploaded by user), query *Cargo IS* database and display a data table with weight and yields of every air route in the uploaded list, as well as a air route map visualizing all the air routes by their cargo weight and cargo yield.

4.1.6.1 Instruction

The idea of yield calculator is to get rid of manual cargo yield computing. To use yield calculator, you will need to create a `csv` file containing two columns: "From" and "To", which correspond to the origin and destination airport. Once the `csv` is ready, you could upload it to **CDM** by clicking **choose file** button. Then, the year parameter should be selected. The others parameters could be modified to adapt the format of your `csv` file. When everything is done, click on *Start*.

A data table will be displayed after the computing. The gross yield, net yield, weight, gross charges and net charges will be calculated for each air route. On the bottom, an air routes map will show the input air routes by the cargo weight and cargo yield. The darker the curve, the higher the yield. The thicker the curve, the higher the cargo weight. Click on "Download" to download the data table. It will be a `csv` file as well.

There may be rows only containing value **zero** in the output table. It is because of the lack of data in Cargo IS database. We do not have any information about the given air route.

On the left of the map, there are some graphical parameters. You can select the appropriate map projection, the comparing criteria and some graphical parameters to customize the map.

4.1.6.2 Explanation

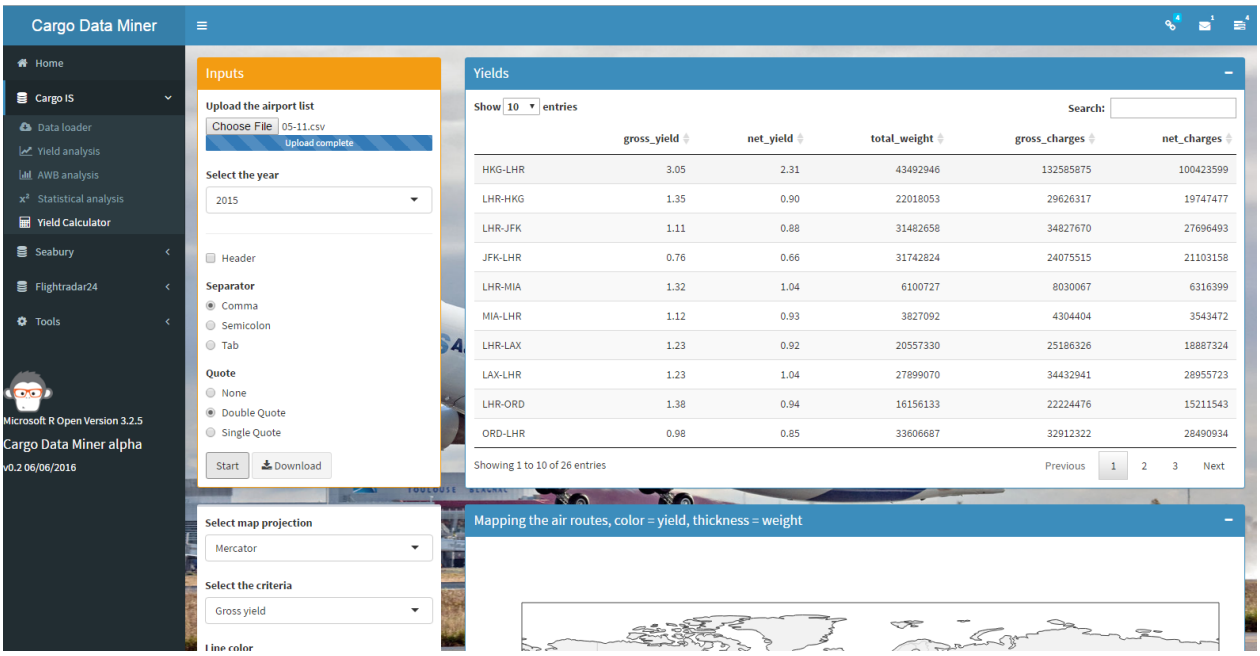

Figure 12: Yield calculator interface

Figure 13 shows the data manipulation procedure. Two input are used. The first one is the uploaded `csv` file and the second one is the Cargo IS data of the selected year (queried by `CargoIS.SQL.query()`). The function `grs_net_yield2()` calls the package `data.table` and computes cargo yield for every air route in this dataset. The yield table will be left joined to the uploaded air route list. After merging, the dataset will be displayed on the top right of the page and the function `Plot.air.route.html()` will be call to further generate the air route map.

4.2 Seabury Database

The second part of this document consists analysis and visualizations of **Seabury Trade** database. Seabury Trade contains annual country level cargo weight and value information. It has not only the airfreight weight and value, but also for the other surface transportation. It provide in-depth information about commodity type which could allow us to get a basical understanding of the local industry pattern.

Beside Seabury trade, there two other databases which could be useful for market knowledge. They are Seabury Express and Seabury. They could be very useful for the further development of the application.

Seabury databases are stored in the same data server as Cargo IS.

4.2.1 Data loading

4.2.1.1 Instruction

As for Cargo IS query, the first step here also consists of data loading. You must select firstly the data granularity to be queried. Then, the year parameter should be set as well. Seabury Trade

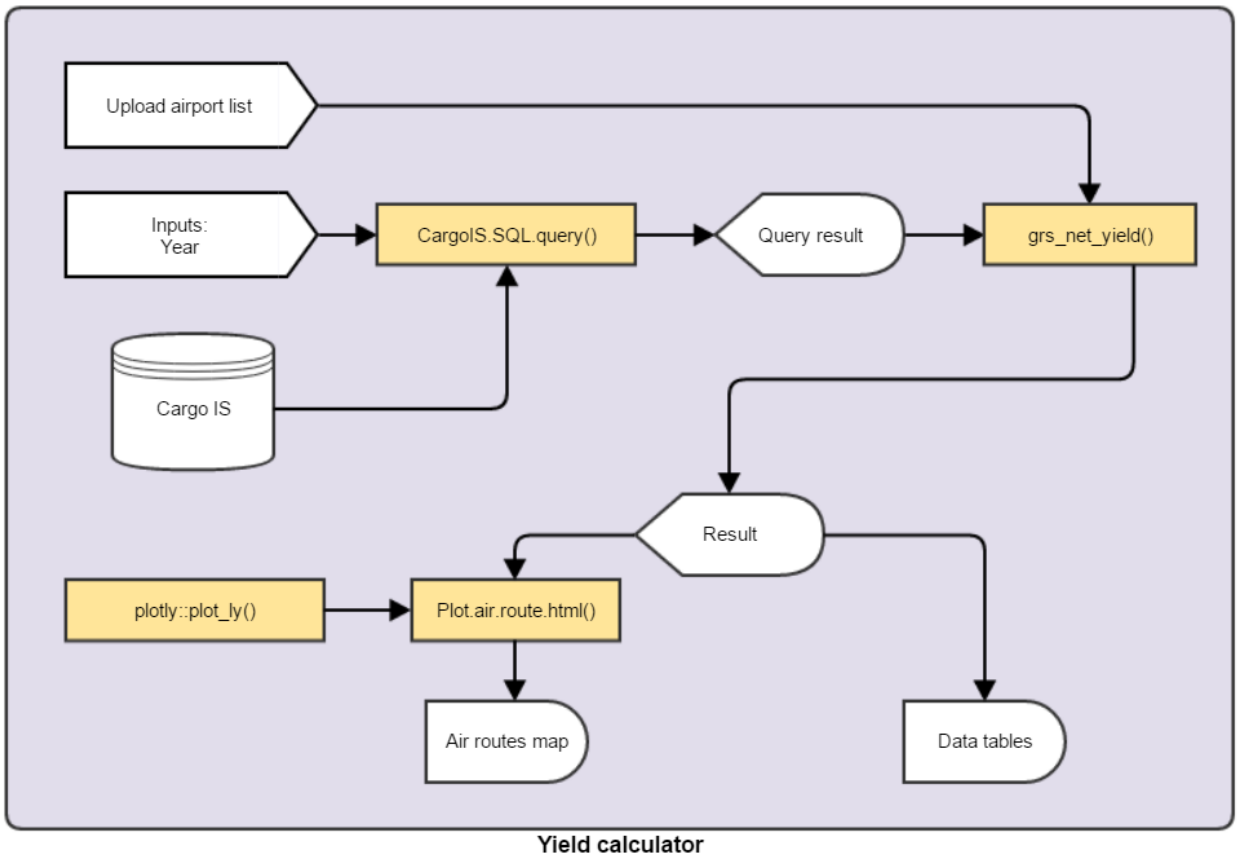

Figure 13: Flowchart of yield calculator

database contains annual data from 2000. Then the drop menus of *Origin* and *Destination* will change automatically according to your selection of data granularity. When all the input parameters are set, click on the **Run** button to start to query the database.

4.2.1.2 Explanation

The start button triggers the user-defined function `Seabury.SQL.OD.query()`. It works just like `CargoIS.SQL.query()`: it will generate a SQL query that depends on the user input. Then it send the generated query to our SQL server and fetch the returned data.

```
# Example
#' ODBC: a Oracle SQL connector,
#' level: granularity of queried data
#' ORG: Origin
#' DST: destination
#' Year: interger of NA, if NA, query all the data, otherwise query the selected year
#' country.name: if TRUE, return the full country name
Seabury.SQL.OD.query(ODBC, level, ORG, DST, year = NA, country.name = FALSE)
```

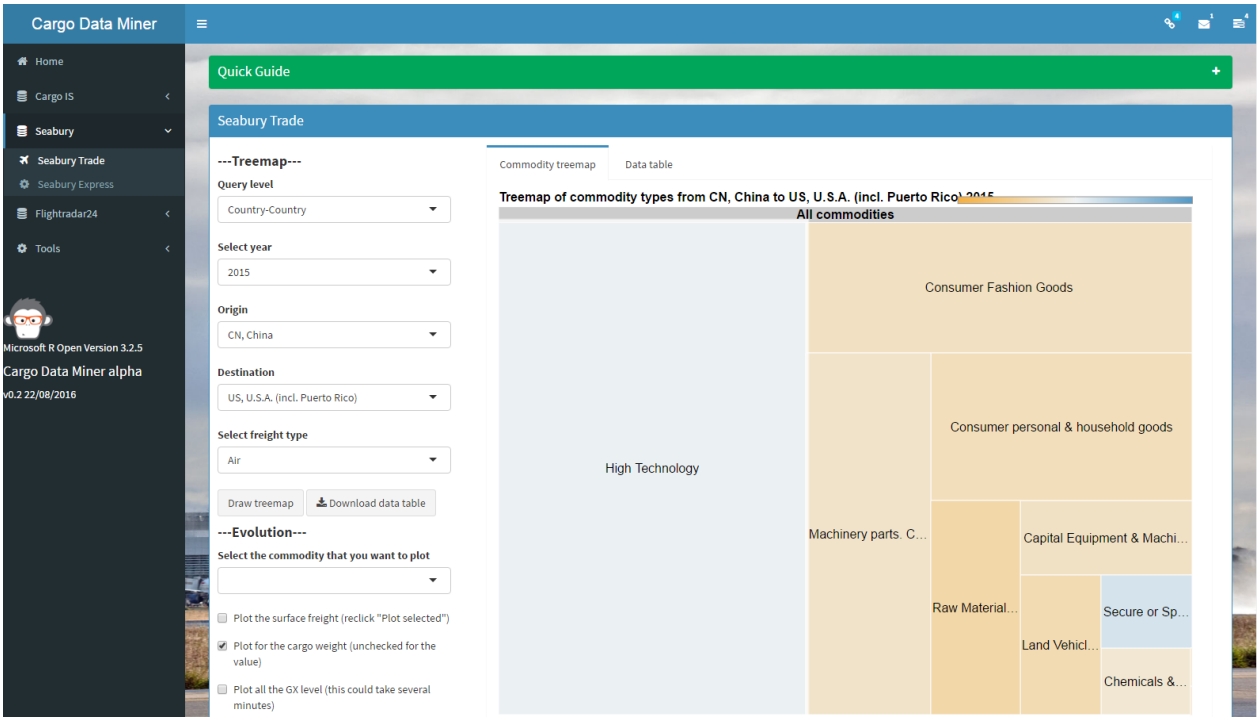

Figure 14: Seabury trade tab

4.2.2 Commodity treemap

Seabury Trade database contains a detailed commodity categorization system. It has eight main categories and more than 2000 sub-categories.

4.2.2.1 Instruction

Seabury Trade has more than 70 000 000 rows so the data loading process may take longer time than Cargo IS. When data is loaded, **CDM** will generate a treemap to present the commodity types, their weight (in kg) and their value (in USD). The size of the square corresponds to the cargo weight for this type of goods. Its color represent the value. The data table could be view in the *Table* tab, and you could download it by clicking on the button *Download*. The structure of the data table is the same as the airport treemap in Cargo IS.

You can click on the square to look further into the sub-categories of the selected commodity type. There are four different levels of categories, G1, G2, G3 and G4. The bigger the number, the higher the complexity. G1 level contains eight different categories but there are around 2000 in the G4 level.

You could change to look at the ocean freight information by simply selecting the freight type from the drop-down menu. The data table is also downloadable by clicking **Download** button.

4.2.2.2 Explanation

To draw the treemap, a user-defined function `treemap_dt_sea_gvis()` is firstly called to generate a data table that fits the requests of `googleVis::gvisTreeMap()`. Then the treemap is generated

with a user-defined function `treemap_sea_gvis()`, which will call `googleVis::gvisTreeMap()`.

```
# Function to generate data table for treemap of SEABURY GXCODE
#' data: an output datatable of Seabury.SQL.OD.query()
#' if AIR = TRUE then output airfreight value
#' otherwise, output surface freight value
treemap_dt_sea_gvis(data, AIR = TRUE)

#' Function to draw the treemap of SEABURY GXCODE
#' data: Output of treemap_dt_sea_gvis()
#' ORG, DST, YEAR: parameter of the title of the treemap
treemap_sea_gvis(data, ORG, DST, YEAR)
```

4.2.3 Evolution plot

We may be able to predict the future by learning from the history. So it will be useful to know how the cargo weight/value of some specific commodities evolve during the last years.

4.2.3.1 Instruction

The second major functionality of Seabury Trade tab is the cargo weight evolution plot. By selecting the name of commodity type, you can thus create a line chart that represents the evolution of airfreight weight since 2000. If the box "Plot all level" is checked, then all the data base will be loaded and visualized (This could take a very long time).

There are four sub-tabs (G1LEVEL, G2LEVEL, G3LEVEL and G4LEVEL) representing the four different categories. The evolution of the sub-categories (if exist) of the selected type will be plotted following sub-tab. You can change the y-axis from cargo weight to cargo value.

The data table will be displayed in the last sub-tab.

Attention: If you selected, for example, a G3 level category, then please go the the third sub-tab (G3LEVEL) to get the corresponded line chart. The charts in the superior levels will be the same as the third tab, but the commodity name will be wrong. To know the level of category that you selected, you can count the number of "-" before the name of category.

4.2.3.2 Explanation

The evolution plot of will use all the Seabury data starting from 2000. The user-defined function `Seabury.SQL.evo.query()` will query the data base and stock the returned data table into the reactive variable `DATA.SEA.GX()`. Then the function `sea.evo()` will be called to sum the weight and value by year and by commodity type (According to the input, it will sum the air freight or the ocean freight data). `Seabury.SQL.get.name()` will return the full name of the commodity types and the names will be added into the data table as a new column. At the end, `Plot_ly()` will be used to create the evolution line chart.

4.3 FlightRadar 24 Database

FlighRadar 24 (FR24) is a recently purchased database of Airbus. FR24 is a Swedish internet-based service that provides aircraft flight information. The database that Airbus purchased includes flight tracks, origins and destinations, flight numbers, registration numbers, flight positions (coordinates), altitude, heading and speed for 2014 and 2015. The data of FR24 mainly comes from the third party ADS-B (Automatic dependent surveillance-broadcast) receivers all over the world. The receivers collect data from any aircraft in their local area that are equipped with an ADS-B transponder and upload to the internet in the real time.

FR24 could be very valuable for Airbus freighter because it may fill the gap of the lack of express cargo airlines' data. In an other flight schedule database **OAG**, there is no data for express cargo airlines (like **UPS**, **FedEx** or **DHL**). However with FR24, we may be able to extract the real flight schedule of these airlines, which we did not have any information before.

But, comparing with other databases, it has some particularities.

- The data size is way larger than the other databases. Because it records the real-time position of aircraft, so we have about 1-2 Gb of data per day. **CDM** mainly uses the air routes data (which means Origin-Destination information). So I could still manipulate the data on my own desktop. But in the future, if we will need to use the position data of FR24, more computing capability will be necessary.
- The data quality of FR24 is a major challenge for analysis. For example here I list three major problems of the database:

  1. Missing values in the data table are very frequent.
  2. There may be several different values indexing one same thing (*e.g.* In the column "Aircraft Type", both B77F, B77L, 77L and 77F represent Boeing 777 freighter).
  3. The data could be merged incorrectly at the very beginning of ADS-B raw data manipulation (*e.g.* the aircraft type of an aircraft registration number is not correct). This could be a problem of FlightRadar24 raw data.

These problems should be solved through two approaches. On one hand, we must return our feedback to FR24 in order to improve the data quality in the raw data level. On the other hand, inside Airbus, we will need to find an efficient method to detect and correct outliers in the database (See the section *Further development* for more details.)

4.3.1 Data loader

Due to the insufficient data quality, the data loading process of FR24 will be a bit more complex than the two previous databases. It will be done in two steps.

First of all, **CDM** queries the FR24 database according to user's input. Then, it will merge the FR24 data with our BIO data by the registration number, in order to improve the data quality.

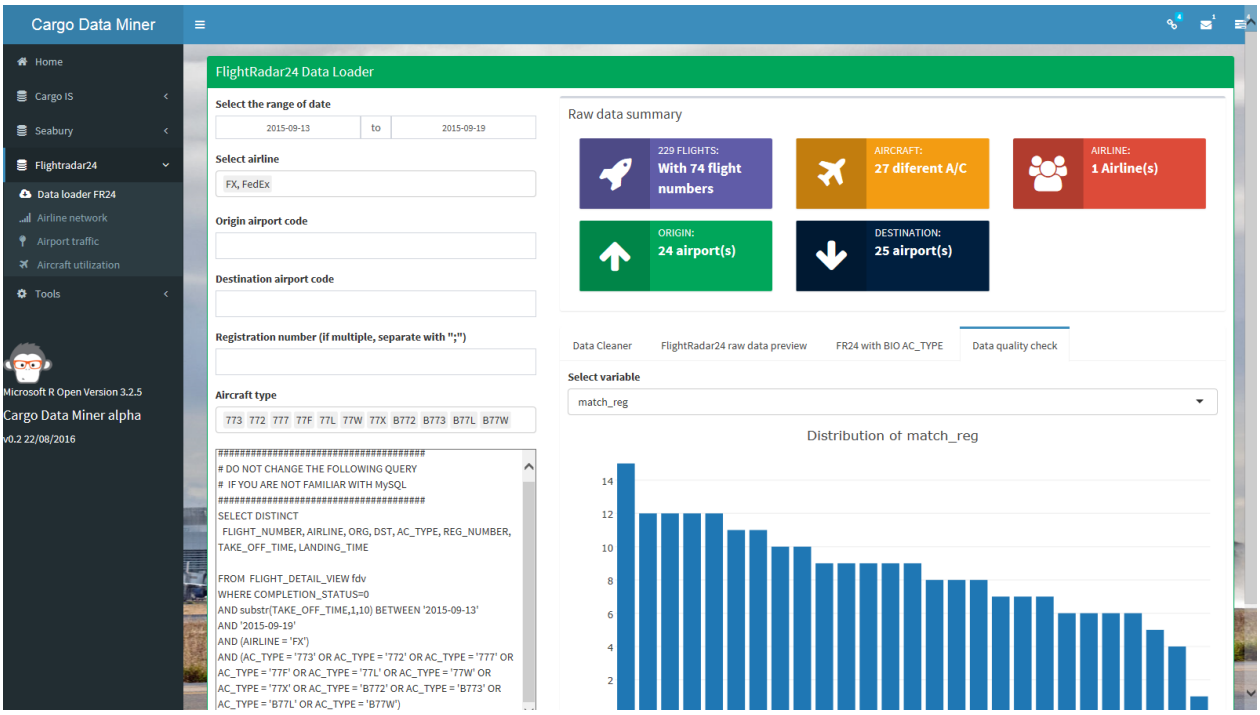

Figure 15: FR24 data loader

4.3.1.1 Instruction

To load the database, you have to precise the date range (by clicking and navigating through the data range selection panel). Then there are several parameters can be set according to the user's need (none of the following parameters is obligatory).

- Airline: Select the airline(s) that you want to query
- Origin: Filter by the origin airport.
- Destination: Filter by the destination airport.
  - If Destination is missing, **CDM** will query all the flights that leave the origin airport
  - If Origin is missing, it will query all the flights that fly to the destination airport
  - If Origin is the same as the Destination, it will query all the flights of the selected airport
  - If both Origin and Destination are missing, it will not filter the data by O&D
- Registration number: Filter by the aircraft registration number(s)
- Aircraft type: Filter by the aircraft type(s)

By modifying the parameters, **CDM** generates automatically a SQL query that corresponds to the input. The query text is showed in the text area. If you are familiar with MySQL, you can directly modify it in the text box to create a more customized query. To solve the problem 1, the query will ignore all the data with a missing origin or destination.

Then click on the button **Query** to query FR24 database. The summary of FR24 data will be showed on the top right of the current tab. Then you can further filter the data by changing the parameters in the box **data cleaner**. By removing the outliers showing in data cleaner, the problem 3 could be solved.

After selecting the appropriated parameters in data cleaner box, click on the blue button **Enrich with BIO** to query the BIO database and merge the BIO data with the FR24 dataset. The idea is to used the cleaned BIO data to replace the original data in order to solve the problem 2. A bar chart will appear on the top right showing the distribution of a selected variable. You could check the data quality again and remove the outliers by looking at its frequency.

The merged data can be downloaded by clicking on the download button.

4.3.1.2 Explanation

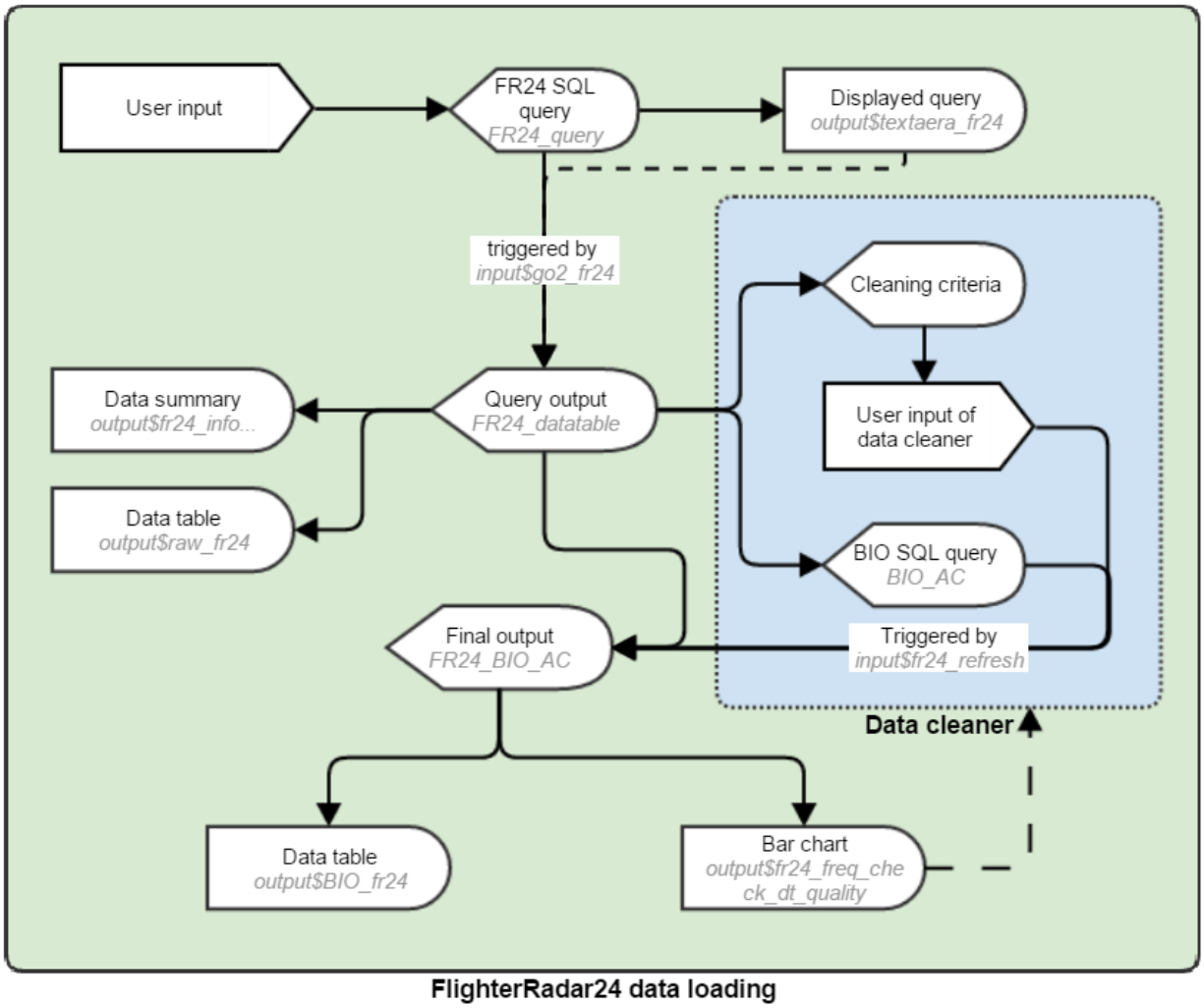

Figure 16: The data loading process of FR24

FR24 is stocked on a MySQL server inside Airbus. So the first step is to generate a SQL query that fits user's request. By using logical operators and string manipulation, the output query string is

stored in `FR24_query()`. The text area shows the content of `FR24_query()` and could be modified by keyboard.

Then, when the button **Query** is clicked, it will create `input$go2_fr24`, which will trigger the SQL query. The query result will be stocked in `FR24_datatable()`. The info-boxes will be created based on descriptive statistics and the preview of data table will be returned as well.

Because of the data quality, I would like to create another filter process to detect the outliers in the data in order to further refine the data quality. The idea is to return the abnormal value in the **Data cleaner** box and let the user decide whether to keep it or not.

- Registration number: The aircraft registrations are strings containing five or six characters. To detect the abnormal value, I used the package `stringdist` to compute the **Jaro-Winkler** distance between each of these numbers. Then I cluster these registration numbers based on the distance matrix. I list the least presented clusters to the user so that the user could select and remove the incorrect values.
- The other three variables contains normally less values than the registration number, so they could by entirely listed and selected by user.

When the button **Enrich with BIO** is clicked, it will trigger the query of the BIO database (**CDM** will extract the BIO data that corresponds to the registration numbers in FR24 dataset) and a dataset containing detailed information of the aircraft will be returned and stored in `FR24_BIO_AC()`. You will see the preview of data table `output$BIO_fr24` and a bar chart `output$fr24_freq_check_dt_quality`.

4.3.2 Airline network

4.3.2.1 Instruction

A map showing the location of every airport in the queried FR24 data table will be displayed after the data loading process. You could click on the markers to see the airport IATA code and airport name. In the tab **Network Analysis**, the network of the data table will be extracted and visualized.

Every circle on the map represents an airport. The size represents its importance in the network. All the circles are colored by their corresponding region. You can filter the airports by their regions by clicking the drop-down menu on the top left.

The arrow shows the direction of the airfreight flow.

You can click on an airport to filter all its neighbor airports (airports that are directly linked to this airport). You can also drag the circle to change its position.

In the third tab analysis, you will see a heatmap showing the distance (minimum numbers of flight) between the airports and a tool to find the shortest path in terms of numbers of flight. You can select the origin and the destination from the two drop-down menus. For the heatmap, the darker the blue, the shorter the path. If the color is white (no matter what the value of z is), it means that there is no flight in between.

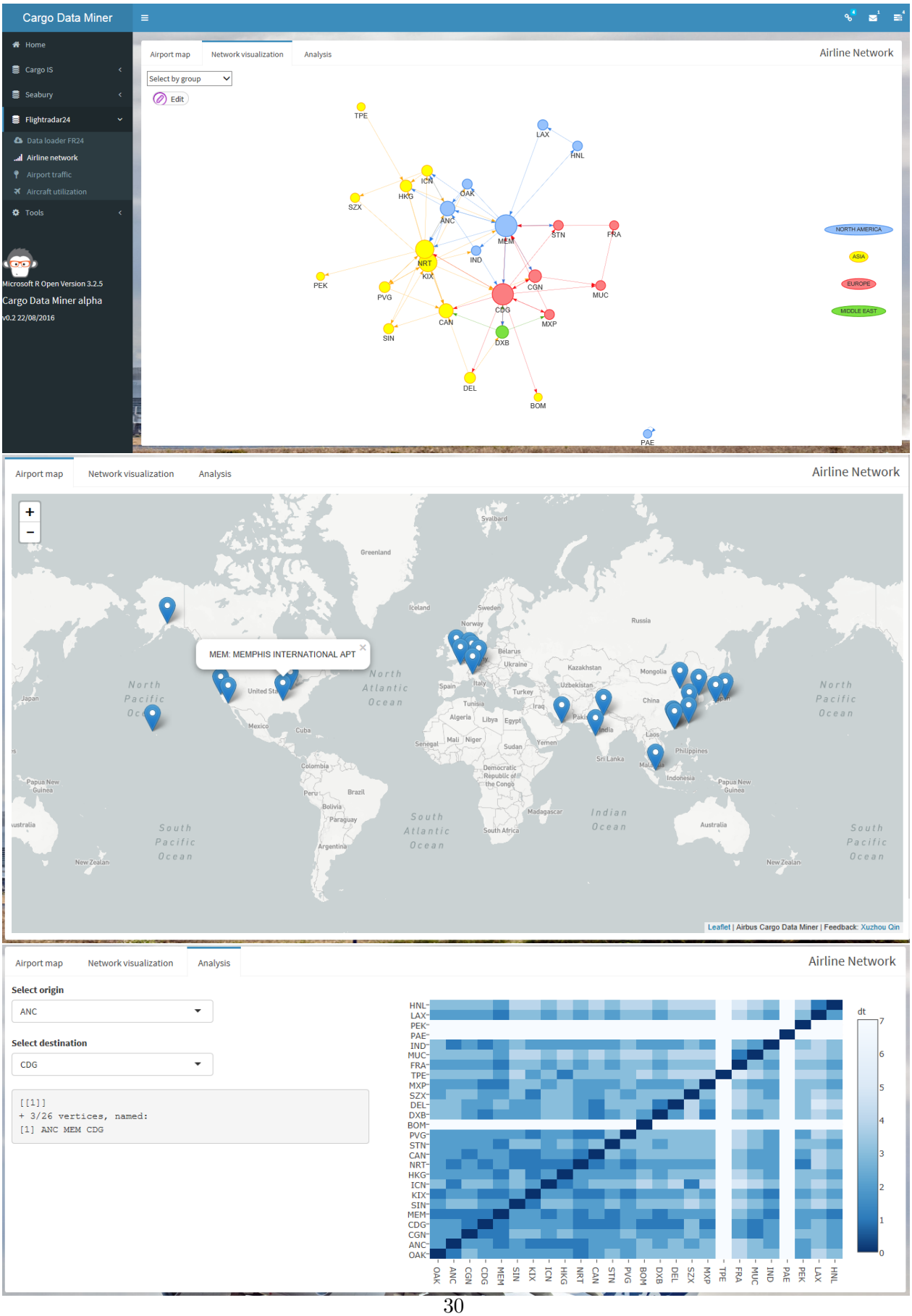

Figure 17: Network visualization for FedEx Boeing 777 Freighter

4.3.2.2 Explanation

Once the reactive value `FR24_BIO_AC()` is created, **CDM** will create `FR24_NW_NODE()`, `FR24_NW_EDGE()` and `FR24_NW_igraph()`. They are three datasets that will be used to generate network visualization.

The airport map is created by using an R package `Leaflet`. It is the API of the open-source JavaScript library `Leaflet` in R. It uses *OpenStreetMap* data to create interactive map. The R package contains only a part of its features, further development could be done by written directly on JavaScript.

```
# EXAMPLE: output$fr24_airline_network_map
# airport is a dataframe containing airports name, longitude and latitude
leaflet(airport) %>%
      addTiles(urlTemplate = "YOUR MAPBOX TEMPLATE",
               attribution = 'YOUR ATTRIBUTION') %>%
      addMarkers(~long, ~lat, popup = ~htmlEscape(paste(IATA,name, sep = ": ")))
```

The network visualization is created by using `visNetwork` package. It takes `FR24_NW_NODE()`, `FR24_NW_EDGE()` as the *nodes* and *edges* parameters.

`FR24_NW_igraph()` is a `igraph` object created from the adjacency matrix (which was created by `igraph::get.adjacency()`). It is a directed, non-weighted adjacency matrix. I used to take the air routes frequency as the weight of edge, but it will sometimes lead to meaningless results. For example in the shortest path computing, if we take into account the frequency of air routes, the shortest path will contain the more "important" air routes instead of the shortest path. For future development, we may add the read great circle distance of the air routes and find a model that determines the weight of air route.

`FR24_NW_igraph()` is thus used for the computing of distance matrix and the heatmap visualization.

4.3.3 Airport traffic*

This tab provides airport traffic information. You will see the take-off rate of the selected airport, the numbers of grounding aircraft, the congestion status, etc.

4.3.4 Aircraft utilization*

In this tab, you can check for each selected aircraft, its utilization (flying time), grounding time and missing time (period that we do not have any information).

You will see the most frequent air routes of the aircraft as well.

4.4 Other features

4.4.1 Connection status check

The first link icon on the right of the header shows the connection status of **CDM** to the databases. If all the databases are successfully connected, the color around the number will be blue. Otherwise the background color will turn red.

You can click on the icon to see the detailed connection information.

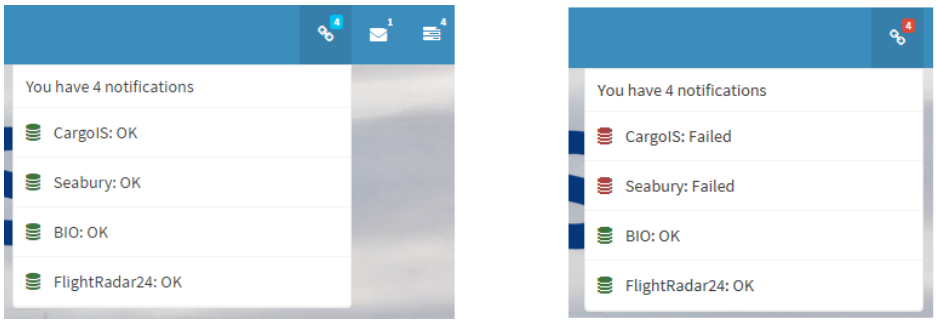

Figure 18: Connection succeed/failed

4.4.1.1 Explanation

Each time when a new session of **CDM** starts, it will try to make a connection to these databases and check the class of the connector by using `try()`. It will return "try error" in the case that the connection of database could not be established.

```
CargoDB <- try(dbConnect(drv, username = username_cargois,
                         password = password_cargois,
                         dbname = connect_string_cargois))
```

If the class of all the connectors are not `'try-error'`, the value `OK` will be given to the variables `statu_cargois`, `statu_bio` and `statu_fr24`, which will determine the color and the text output of the header's notification.

IMPORTANT: Functions that related to a database will be loaded only if the database could be reached.

4.4.2 Server-side selectize input

There are several different methods to have a reactive select input. For example `uiOutput() + renderUI({selectInput()})`. But in some situation, the input contains too many elements so that the performance of `renderUI()` may be insufficient. In this situation, `selectizeInput() + updateSelectizeInput()` will be very useful.

```
# Example
# in ui.R
selectizeInput('foo', choices = NULL, ...)

# in server.R
shinyServer(function(input, output, session) {
  updateSelectizeInput(session, 'foo', choices = data, server = TRUE)
})
```

The server-side `selectizeInput` uses R to process searching, and R will return the filtered data to `selectize`. It is widely use in **CDM** and it greatly boosts the interactive performance.

4.4.3 User session log (still under development)

The idea of this feature is to create a interface where the experienced R user could get the system log information and interactive with R by coding. But now I still did not find a way to capture the R console's output and return it into **CDM**. Now you can only call a "so-called" console of **CDM** by pressing the ~ button of your keyboard. It will display your detailed connection status.

This feature is done by using a JS script.

```
# ui.R
fluidRow(
         htmlOutput("consol"),
         tags$script('
                     $(document).on("keypress", function (e) {
                     Shiny.onInputChange("key_track", e.which);
                     });
         ')
)

# server.R
output$consol <- renderUI({
    show = FALSE
    if(is.null(input$key_track)){
    } else if (input$key_track == 126) {
      show <- TRUE
    }
    if(show){
      HTML('<!-- HTML CODE HERE -->')
    } else {
      p('')
    }
  })
```

5 Future Development

Several functionalities and features of **CDM** is still under development. Some existing features could be improved and new functionalities could be added.

5.1 Improvement of existing features

5.1.1 Speed up data processing

- Some functions still use `data.frame` objects to manipulate data. In order to have a faster processing speed, the package `data.table` should be used extensively.
- Avoid using `cbind` to expanding existing data structures. Try always to initialize it and then fill them in.
- Store the results and access them rather than repeatedly recalculating.

- Instead of using loops, try to use `apply(), sapply(), lapply()`.
- Try to use multiple-thread computing (with MRO or packages like `parallel`)

5.1.2 Downloadable images

All the `ggplot2` charts could be downloaded. However, for the charts created with `googleVis`, we will need to find a solution to generate a static image that could be downloaded. For those which are generated by `plot_ly`, figure out the problem that sometimes the button *download* does not work.

5.1.3 Database connections

Optimize the control of existing connection, automatically close those which are not used any more to gain resources.

5.1.4 Data cleaner of FR24

Create a more accurate algorithm to detect the outliers of the data. Then, instead of using `selectizeInput()`, try to find a more pretty way to interact and filter the data.

Optimize the data loading process of FR24 to make it more intuitive. Now two clicks are needed to generate the dataset. In the future, we may try to improve the user's experience and reduce the number of clicks.

5.1.5 Personalized error message

In some rare situation, **CDM** will stop running and crash if the data structure could not be created properly (for example, if the query result is empty). In these cases, render a personalized error message in order to avoid the crash of application.

5.1.6 Color of the treemap and circle size of top airport map

Find a algorithm to allocate a more appropriate value that determines the color of squares in the treemaps and the size of circle in airport map.

Then, try to make the color of treemap horizontally comparable. This means that you will be able to compare the color in different treemaps. Now you can only compare the color within on treemap.

5.1.7 Modulize the application code

Inspired by the article Modularizing Shiny app code, When **CDM** grows larger and more complicated, the complexity of code grows as well.

Modules should be used in order to make the code maintenance easier.

5.2 New features to be added in the future

5.2.1 A general chart creating tool

The idea is to make the data visualization power of R be usable for everyone. To do so, I would like to create a visualization tool that could generate a customizable chart from a user uploaded `csv` or `xlsx` file. The file contents could be varied.

5.2.2 Standardise the definitions of variable

In **CDM**, the definitions of variable may vary according to databases. For example the definition of *region* in **Cargo IS** and **Seabury**. If we could have a standard and unique definition within **CDM**, then it will be possible to merge these databases and increase the dimension of data.

5.2.3 Automated reports generator

Inspired by project *LiMA*, a report generator could be used for automatic generation of a `pdf`, `doc` or `html` file, which contains all the analysis results created in **CDM** (or even from an uploaded `xlxs` file). The R package `knitr` could be a good choice to do this task.

6 Conclusion

Cargo Data Miner will be able to replace our three standalone database query tools with one single interactive interface providing access to all three databases. Aside of time saving for accessing and analyzing the data, **CDM** also allows us to link the three databases together for a more sophisticated and detailed analysis.

R Shiny Dashboard provides thus a solution of a light-weight and customizable data analysis/visualization tool. It turned out to be a good choice to fit the requests of the Airbus Freighter Marketing team. **CDM** is the first step in the Airbus freighter department as the response to the company level movement of digitalization.

In this Digital Age, the revolutionary technologies opened the door of large scale sharing and collaborating. Not only you can complete a task more efficiently and accurately, but you can also share the results instantly with your team and across the company. Digitalization **is not** only about having all the data, it is also about the way we handle our data and our business.

Interaction is the key word of this age. It gives faster and easier exchanges of information, better adaptability, smarter business intelligence and most importantly, higher end-user productivity. All these could produce large benefits for our business. Airbus is just at the beginning of data digitalization. In order to keep us competitive, challenges still reside.

The following are the domains that matter the most from my point of view:

- **Cloud**: It is definitely the most important part of digitalization. It provides shared computer processing resources and data services, which could be easily and optimally allocated to user according to the demand.

- **Collaboration**: We need a platform to share and to work together simultaneously. It should be possible for every employee to share their own project on the platform and allow others to become contributors of it.
- **Big data**: Not only the quantity of data, but also the dimension of data is increasing exponentially. The data may seem very sparse and irrelevant to our business. New approach should be adopted to extract the maximum value from the data.

I am thrilled to see that some initial attempts are already realized in Airbus. Not only **CDM**, other Shiny Dashboard application have been created as well. We also have a newly built platform **Foundry**, where we could use SparkR to work on the collective data, share our codes and contribute to other colleagues' project.

If we always stay open-mind with new technologies and keep our infrastructure up to data, I believe Airbus will be able to reach a higher goal in the Digital Age.

7 Appendix I: Troubleshooting

7.1 Install Oracle Database instance client

Attention: You will need a local **TA** account before starting the manual installation.

Oracle Database Instant Client (11.2.0.3) EN_W764 could be found in **PC Service**, or it could be downloaded from http://www.oracle.com/technetwork/topics/winx64soft-089540.html (you may need to create an account to be able to download it). After downloading the client, extract it to a folder that you could find again.

If **CDM** could not be executed due to errors of Oracle instant client, you can check if the environment variables of Windows is set correctly by doing the following procedure.

- Open the properties window of the computer.
- Click on "**Advanced system settings**"
- Select the tab "**Advanced**" of the appearing dialog box.
- Click "**Environment Variables**" button on the bottom.
- Find the variable "**Path**", make sure that the path of your Oracle instance client is added. If not, add it.

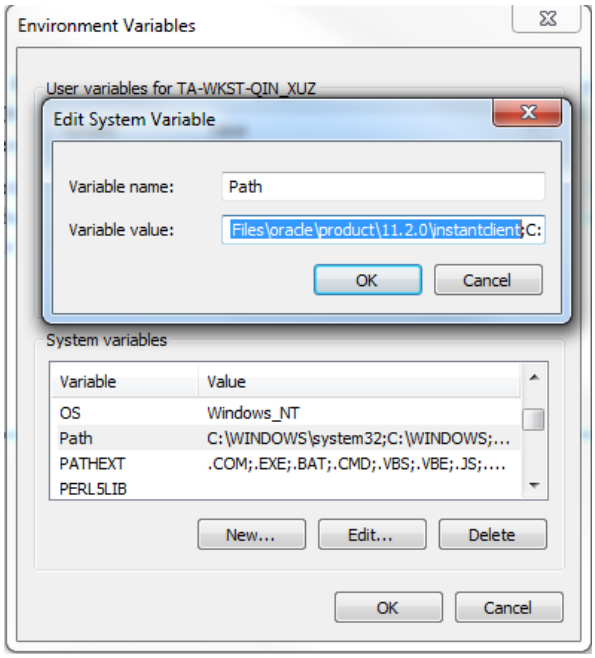

Figure 19: Check the "Path" system variable

If the problem still exists, check the **ROracle** package installation instruction on http://cran.us.r-project.org/web/packages/ROracle/INSTALL

7.2 Download and install an R package manually

Some of the R package could not be installed directly from **CRAN**, for example the package `ROracle`. In this situation, the manual installation will be needed.

Here I will show the installation process for `ROracle`.

- Firstly, download `ROracle_1.2-1.zip` from *oracle.com* (http://www.oracle.com/technetwork/database/databa technologies/r/roracle/downloads/index.html)
- Secondly, place the .zip file into `~/CDM/Rpackages` and make sure that *Oracle Instant Client* is correctly installed.
- Then, instead of running

```
install.packages('ROracle')
```

run

```
install.packages('Rpackages/ROracle_1.2-1.zip', repos = NULL, type="source")
```

After the installation, the package could be called by

```
library(ROracle)
```

8 Appendix II

8.1 Cargo IS countries

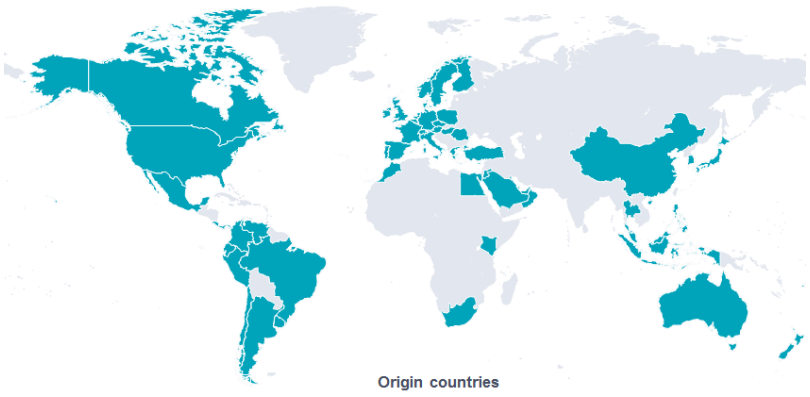

Figure 20: Origin countries in Cargo IS database

9 Bibliography

- Patrick Burns, *The R Inferno*, http://www.burns-stat.com/pages/Tutor/R_inferno.pdf
- Navarro, Gonzalo (2001), *A guided tour to approximating string matching*, ACM Computing Surveys. 33 (1): 31–88. doi:10.1145/375360.375365

- `ggplot2` documents, http://docs.ggplot2.org/current/
- `Plot_ly` documents, https://plot.ly/r/
- `Google Chart` documents, https://developers.google.com/chart/interactive/docs/
- `leaflet` documents, http://leafletjs.com/reference.html
- `R Shiny` articles, http://shiny.rstudio.com/articles/
- Katherine Ognyanova, *Network visualization with R*, http://kateto.net/network-visualization, POLNET 2016 Workshop, St. Louis, MO
- Jon Duckett, *HTML and CSS: Design and Build Web Sites*